**EASTERN MEDITERRANEAN UNIVERSITY**

**DEPARTMENT OF COMPUTER ENGINEERING**

**CMSE 443**

**Real-Time Systems Design**

**Final Exam**

**2023/24 Fall Semester**

**January 8, 2024**

**Name-Surname : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Student Number : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Instructor:

Prof. Dr. Alexander CHEFRANOV

Duration: 130 minutes

Mobiles, calculators, and other electronic devices ARE NOT ALLOWED

YOU CAN BRING FOUR A4 SIZED SHEETS OF *HAND-WRITTEN* NOTES TO THE EXAM. PHOTOCOPIES ARE NOT ALLOWED AND WILL BE COLLECTED.

TRY ALL THE QUESTIONS

**Totally 6 questions, 13 pages**

**Section I. RTS Hardware (33 points)**

[**T2**](#T2)[**T3**](#T3)[**T4**](#T4)[**T5**](#T5)[**T6**](#T6)

**[table](#table)Task 1 (17 points)** For the expression

z=(a+b-c\*d)/(x+y)

1. **(4 points)** Build the tree representation of the expression
2. **(4 points)** Build the reverse Polish notation of the expression using by LRN (left-right-node) tree traversal strategy

Zab+cd\*-xy+/=

1. **(4 points)** Write the code using instructions of 0-address machine to implement the expression using its reverse Polish notation
2. Load\_literal @z
3. Load literal @a
4. Load
5. Load\_literal @b
6. Load
7. Add
8. Load\_literal @c
9. Load
10. Load\_literal @d
11. Load
12. Mul
13. Sub
14. Load\_literal @x
15. Load
16. Load\_literal @y
17. Load
18. add
19. div
20. sto
21. **(5 points)** Assume that addresses and values of the variables are as follows

|  |  |  |  |
| --- | --- | --- | --- |
| # | Name | Address | Value |
| 1 | a | 100 | 10 |
| 2 | B | 101 | 12 |
| 3 | C | 102 | 3 |
| 4 | D | 103 | 4 |
| 5 | X | 104 | 2 |
| 6 | y | 105 | 3 |
| 7 | z | 106 | 1 |

Calculate the expression using the code generated showing all intermediate variables’ values and content of the stack (growing horizontally from left to right, e.g. a three-element stack is represented by (100, 25, 11) with 11 on the top and 100 on the bottom) in the following table:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| # | Instruction | A (100) | B (101) | C (102) | D (103) | X (104) | Y (105) | Z (106) | Stack content |
|  | Load\_literal @z | 10 | 12 | 3 | 4 | 2 | 3 | 1 | 106 |
|  | Load literal @a |  |  |  |  |  |  |  | 106, 100 |
|  | Load |  |  |  |  |  |  |  | 106, 10 |
|  | Load\_literal @b |  |  |  |  |  |  |  | 106, 10, 101 |
|  | Load |  |  |  |  |  |  |  | 106, 10, 12 |
|  | Add |  |  |  |  |  |  |  | 106, 22 |
|  | Load\_literal @c |  |  |  |  |  |  |  | 106, 22, 102 |
|  | Load |  |  |  |  |  |  |  | 106, 22, 3 |
|  | Load\_literal @d |  |  |  |  |  |  |  | 106, 22, 3, 103 |
|  | Load |  |  |  |  |  |  |  | 106, 22, 3, 4 |
|  | Mul |  |  |  |  |  |  |  | 106, 22, 12 |
|  | Sub |  |  |  |  |  |  |  | 106, 10 |
|  | Load\_literal @x |  |  |  |  |  |  |  | 106, 10, 104 |
|  | Load |  |  |  |  |  |  |  | 106, 10, 2 |
|  | Load\_literal @y |  |  |  |  |  |  |  | 106, 10, 2, 105 |
|  | Load |  |  |  |  |  |  |  | 106, 10, 2, 3 |
|  | add |  |  |  |  |  |  |  | 106, 10, 5 |
|  | div |  |  |  |  |  |  |  | 106, 2 |
|  | sto |  |  |  |  |  |  | 2 |  |
|  |  |  |  |  |  |  |  |  |  |

**Hints**:

0-address machine instruction set is as follows:

|  |  |
| --- | --- |
| **0-address machine (reverse Polish)** | |
| where the processor has a stack and some supporting hardware, at least a top of stack (TOS) pointer. | |
| **Operation** | **e.g. or comment** |
| load\_literal <int> | effect: TOS:=TOS+1; stack[TOS]:=<int> load a *constant* onto the top of stack; this can be used in arithmetic or to get an address onto the stack for use by a load or a store instruction later (it is splitting hairs to argue whether the literal is an address or a constant which might happen to be used as an address elsewhere) |
| load | effect: stack[TOS]:=memory[stack[TOS]] take the top-of-stack as an address, replace the top-of-stack with the contents of that address. |
| sto | effect: memory[stack[TOS-1]]:=stack[TOS]; TOS:=TOS-2 store contents of top of stack at the address in stack[TOS-1] then pop the value and the address |
| <opcd> | where <opcd> is add | sub |... effect: stack[TOS-1] := stack[TOS-1] <op> stack[TOS]; TOS:=TOS-1 |

**[T1](#T1)Task 2 (16 points)**

1. **(6 points)**  Fill in the table below by the binary equivalents of the following five decimal numbers: 13, 11, 7, 15, 25,6

|  |  |
| --- | --- |
| Decimal Number | Binary number |
| 13 | 01101 |
| 11 | 01011 |
| 7 | 00111 |
| 15 | 01111 |
| 25 | 11001 |
| 6 | 00110 |

1. **(10 points)** Draw a PLA schema programmed by fusing (connecting by dot) lines to implement a ROM with 6 memory cells containing the numbers in the above table in the given order.

Truth table for the ROM

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Address | A2 | A1 | A0 | X4 | X3 | X2 | X1 | X0 | Number |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 13 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 11 |
| 2 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 7 |
| 3 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 15 |
| 4 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 25 |
| 5 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 6 |

a2

a2

&0

&1

&2

&3

a1

a1

a0

a0

V

V

V

V

V

&4

X4

X3

X2

X1

X0

&5

**Hints**: Example of PLA with 2 inputs and 4 outputs



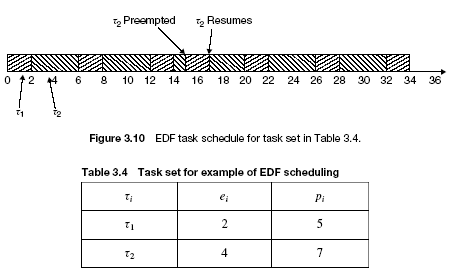
**Section II. RTS Operating Systems (67 points)**

**[T1](#T1)Task 3.** **(16 points)** Build an Earliest-Deadline First schedule for two periodic tasks:

|  |  |  |
| --- | --- | --- |
| Task# | E | P |
| 1 | 3 | 6 |
| 2 | 2 | 4 |

What are the utilization and hyperperiod for this set of tasks? Are the deadlines met by the RM schedule? Explain why they are met (not met)?

**Hints:**

****

Hyperperiod

Utilization is . Since phase and relative deadline of the tasks are not specified, we take by default the first release time is zero, and the relative deadline is equal to the period. Construct EDF schedule for H time units:

2

2

1

1

1

2

2

1

1

1

2

2

Define a table showing release time, deadline, and completion time for each task release according to the EDF schedule constructed:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | task | release | Release time | Deadline | Completion time |
| 1 | 1 | 1 | 0 | 6 | 5 |
| 2 |  | 2 | 6 | 12 | 10 |
| 3 | 2 | 1 | 0 | 4 | 2 |
| 4 |  | 2 | 4 | 8 | 7 |
| 5 |  | 3 | 8 | 12 | 12 |

It is seen from the above table that there is no any task release completing after its deadline, hence, all the deadlines are met.

**[T1](#T1)Task 4.** **(17 points)** Assume, four parallel processes, A, B, C, and D, run concurrently with the dependency graph as follows:

Processes B and C wait for a data from A, process it, and provide their outputs to D waiting for them. Process A waits for the signal from D. Write a pseudo-code using binary semaphores to provide necessary synchronization of the processes A, B, C, and D. Define necessary data structures. Show initial settings of the global semaphores you use.

A

B

C

D

**Hints**:

Driver{ while(1){

if(data\_for\_I/O){

prepare(command);

V(busy); P(done);}

}

}

Controller{while(1){

P(busy);

exec(command);

V(done);

}

}

|  |  |  |  |
| --- | --- | --- | --- |
| Binary\_Semaphore semABdata=0, semACdata=0,semBdata=0, semCdata=0, semD=0;  Data Adata, Bdata, Cdata;//Data is some predefined data type | | | |
| Process A(){  While(1){  Prepare(Adata);  V(semABdata);  V(semACdata);  P(semD);  }  } | Process B(){  While(1){  P(semABdata);  Prepare(Bdata);  V(semBdata);  }  } | Process C(){  While(1){  P(semACdata);  Prepare(Cdata);  V(semCdata);  }  } | Process D(){  While(1){  P(semBdata);  P(semCdata);  V(semD);  }  } |

**[T1](#T1)Task 5. (17 points)** Assume that a system has 5 processes and resources of two types: processors (totally available 8) and memory (total available number of memory blocks is 10). Processes’ resources required and maximal requirements are as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Process | Processors required | Processors maximal required | Memory required | Memory maximal required |
| 1 | 2 | 4 | 3 | 6 |
| 2 | 2 | 6 | 2 | 5 |
| 3 | 1 | 5 | 1 | 4 |
| 4 | 1 | 4 | 2 | 3 |
| 5 | 1 | 6 | 1 | 5 |

Use the Banker’s algorithm to decide on the safety of granting the required resources. Show all the steps of the algorithm you make to come to your decision.

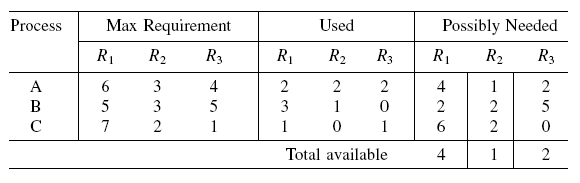
**Hints**:

**The Banker’s Algorithm**

The case of multiple resources. Initial resource state:



Safe state with safe sequence A, C, B:



Let’s extend our table by showing the possibly needed requests and totally available after granting resources:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Process | Processors required | Processors maximal required | Memory required | Memory maximal required | Possibly  needed processors | Possibly needed memory |
| 1 | 2 | 4 | 3 | 6 | 2 | 3 |
| 2 | 2 | 6 | 2 | 5 | 4 | 3 |
| 3 | 1 | 5 | 1 | 4 | 4 | 3 |
| 4 | 1 | 4 | 2 | 3 | 3 | 1 |
| 5 | 1 | 6 | 1 | 5 | 5 | 4 |
| sum | 7 |  | 9 | Available after granting | 8-7=1 | 10-9=1 |

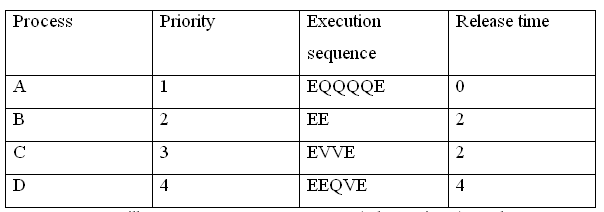
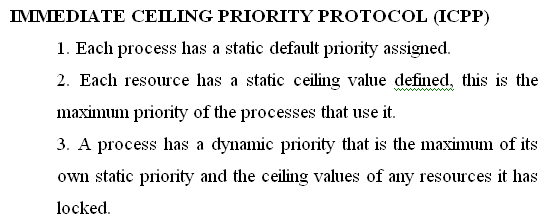
To decide on the safety of the resources granting, the Banker’s algorithm tries building a sequence of the processes such that each process in the sequence can get its maximal required resources and terminate. If such a sequence including all the processes is found then the state is safe and the resources required may be granted. Otherwise, the state is not safe, and the resources required are not granted. Since only 1 processor and 1 memory block are left, no one process’ possibly needed request can be satisfied, and, hence, the safe sequence does not exist. Hence, such a state is not safe, and the resources required are not to be granted.

**[T1](#T1)Task 6. (17 points)** Assume, we have a priority preemptive system (without time sharing for same priority tasks) and the following system of processes:

|  |  |  |  |
| --- | --- | --- | --- |
| Process | Priority | Execution sequence | Release time |
| A | 2 | QVQVVEV | 2 |
| B | 4 (highest) | EQEQQ | 3 |
| C | 1 (lowest) | EQQQVVE | 1 |
| D | 3 | QVVVQEVV | 0 |

Show time diagrams of execution for each process, if Immediate Ceiling Priority Protocol is used. Calculate response time for each process A, B, C, D. Specify dynamic priority for each task and each time unit when it runs.

**Hints:**

** **

****

A

P

P

P

P

P

P

P

P

P

P

B

E

Q

E

Q

Q

C

P

P

P

P

P

P

P

P

P

P

P

D

Q

V

V

P

P

P

P

P

V

Q

E

V

V

P

P

A

Q

V

Q

V

V

E

V

C

P

P

P

P

P

P

P

E

Q

Q

Q

V

V

E

Cont.

|  |  |  |  |
| --- | --- | --- | --- |
| Process | Release time | Completion time | Response time |
| A | 2 | 20 | 18 |
| B | 3 | 8 | 5 |
| C | 1 | 27 | 26 |
| D | 0 | 13 | 13 |

Ceil(Q)=4; ceil(V)=3

Process A runs on its own priority 2 when just executing (E), on priority 4 when using Q, and on priority 3 when using V.

Process B runs on its own priority 4.

Process C runs on its own priority 1 when just executing (E), on priority 4 when using Q, and on priority 3 when using V.

Process D runs on its own priority 3 when just executing (E) and using V, and on priority 4 when using Q.

[T1](#T1) [T2](#T2) [T3](#T3) [T4](#T4) [T5](#T5) [T6](#T6)